Co-funded by the
European Union

INTERNET I RAČUNARSKA SIGURNOST

Internet and computer security

Bihać, 2025

POPIS SLIKA

SADRŽAJ

UVOD

Internet i računarska sigurnost su ključni faktori za sigurnost digitalnog svijeta u kojem živimo. Kako tehnologija napreduje, tako raste i potreba za adekvatnom zaštitom računarskih sistema, podataka i korisničkih informacija na internetu. U ovom kursu, kroz detaljnu teoretsku osnovu i praktične primjere, obradit će se ključne teme poput zaštite računara na internetu, sigurnosnih prijetnji, metoda zaštite, te kriptografije, koja je osnova sigurnosti podataka i komunikacija. Posebnu pažnju posvetit će se primjeni Pythona u implementaciji zaštitnih metoda, kao i analizi i implementaciji različitih kriptosistema, kao što su simetrična i asimetrična kriptografija, hash funkcije i generatori slučajnih brojeva. Kurs je osmišljen kako bi polaznicima omogućio razumijevanje teorijskih osnova i praktičnih veština potrebnih za upravljanje sigurnošću u digitalnom okruženju.

1. INTERNET

1.1. Uvod u internet

Internet je globalna mreža računara koja omogućava međusobnu komunikaciju i razmjenu podataka koristeći različite tehnologije i protokole. Nastao je kao rezultat razvoja računarskih mreža i danas predstavlja osnovu modernog digitalnog svijeta. Milijarde uređaja su povezane putem interneta, a njegova uloga je prisutna u gotovo svim sferama ljudskog života – od obrazovanja i poslovanja do zabave i društvenih interakcija.

Internet funkcioniše na osnovu niza pravila i standardizovanih protokola koji omogućavaju komunikaciju između računara. Ključni elementi interneta uključuju:

- **IP adrese** *(eng. Internet Protocol Address)* – Svaki uređaj povezan na internet ima jedinstvenu numeričku oznaku koja mu omogućava identifikaciju i komunikaciju s drugim uređajima.

- **DNS** *(eng. Domain Name System)* – Servis koji prevodi ljudima razumljive web adrese (npr. www.google.com) u numeričke IP adrese koje računari koriste za komunikaciju.

- **Protokoli za prijenos podataka** – Pravila koja omogućavaju sigurnu i pouzdanu razmjenu informacija (HTTP, HTTPS, FTP, TCP/IP, SMTP, IMAP, itd.).

- **Servisi na internetu** – Web stranice, društvene mreže, e-mail servisi, cloud usluge, e-trgovina, online bankarstvo i mnogi drugi oblici digitalnih interakcija.

Međutim, upotreba interneta nosi i određene sigurnosne rizike. Zbog toga je važno razumjeti metode zaštite koje omogućavaju sigurno korištenje interneta.

1.2. Načini zaštite računara i korisnika na internetu

Prilikom korištenja interneta, korisnici su izloženi različitim prijetnjama koje mogu ugroziti njihovu privatnost, sigurnost podataka i stabilnost sistema. Napadači koriste razne tehnike za krađu informacija, manipulaciju korisnicima ili oštećenje računarskih sistema.

U nastavku će se detaljno objasniti glavne prijetnje na internetu i metode zaštite koje korisnici mogu primijeniti.

1.2.1. Sigurnosne prijetnje na internetu

Malveri (Malware) – virusi, trojanci, ransomware

Malveri su zlonamjerni softveri dizajnirani s ciljem narušavanja sigurnosti korisnika. Najčešće vrste malvera su:

- **Virusi** - Računarski programi koji se prikače na druge legitimne programe i izvršavaju štetne radnje.

- **Trojanci** - Maliciozni softveri koji se prikrivaju kao korisni programi, ali zapravo omogućavaju hakerima pristup sistemu.

- **Ransomware** – Malver koji šifrira podatke na računaru i zahtijeva otkupninu za njihovo vraćanje.

Praktični savjeti:

- Preuzimati softver samo s provjerenih izvora.
- Koristiti pouzdane antivirusne programe kao što su Windows Defender, Avast, Kaspersky ili Bitdefender.
- Redovno skenirati računar i ažurirati sigurnosni softver.

Phishing i socijalni inženjering

Phishing napadi se zasnivaju na prevari korisnika pomoću lažnih e-mailova ili web stranica koje oponašaju legitimne servise. Cilj je navesti korisnika da unese svoje povjerljive podatke, kao što su lozinke i brojevi kreditnih kartica.

Praktični savjeti:
- Provjeriti e-mail adresu pošiljaoca i ne klikati na sumnjive linkove.
- Koristiti dvofaktorsku autentifikaciju (2FA) za dodatni nivo zaštite.
- Redovno mijenjati lozinke i koristiti menadžere lozinki.

Napadi na mreže (DDoS, MITM)

- **DDoS napadi** – Napadači koriste mreže zaraženih računara (botnet) kako bi preopteretili servere velikim brojem zahtjeva i učinili ih nedostupnim.

- **MITM** *(eng. Man-in-the-Middle)* **napadi** – Napadači presreću komunikaciju između korisnika i web servisa, čime mogu ukrasti podatke ili manipulisati informacijama.

Praktični savjeti:

- Koristiti VPN servise za sigurnu enkripciju mrežnog saobraćaja.
- Omogućiti HTTPS protokol prilikom pristupa osjetljivim web stranicama.
- Izbjegavati povezivanje na nesigurne Wi-Fi mreže.

1.2.2. Metode zaštite na internetu

Antivirusni softver i vatrozid

Antivirusni programi su ključna zaštita od malvera, dok vatrozid filtrira mrežni saobraćaj i sprečava neovlaštene pristupe.

Primjer: Windows Defender aktivacija

```
Settings → Update & Security → Windows Security → Virus & threat protection
```

Sigurnosne prakse prilikom korištenja e-maila i društvenih mreža:

- Ne otvarati nepoznate e-mail priloge i linkove.
- Aktivirati dvofaktorsku autentifikaciju (2FA).

Primjer: Gmail 2FA aktivacija

```
Google Account Settings → Security → 2-Step Verification
```

Enkripcija podataka i VPN

VPN servisi pružaju dodatnu sigurnost šifriranjem internet saobraćaja i sakrivanjem IP adrese korisnika.

Primjer: Instalacija Windscribe VPN-a

```
https://windscribe.com/
```

1.3. Primjer zaštite na internetu pomoću Pythona

Python se može koristiti za zaštitu na internetu pomoću različitih sigurnosnih tehnika, kao što su provjera URL-a na phishing i šifriranje podataka.

Provjera sigurnosti web stranice pomoću Python skripte

Ova skripta koristi *VirusTotal API* za provjeru da li je web stranica maliciozna:

```
import requests

API_KEY = "YOUR_VIRUSTOTAL_API_KEY"
url_to_check = "http://example.com"

def check_url_safety(url):
    params = {"apikey": API_KEY, "resource": url}
    response = requests.get("https://www.virustotal.com/vtapi/v2/url/report",
params=params)

    if response.status_code == 200:
        result = response.json()
        if result["positives"] > 0:
            print(f"Upozorenje: {url} je označen kao nesiguran!")
        else:
            print(f"{url} je siguran za posjetu.")
    else:
        print("Greška pri provjeri URL-a.")

check_url_safety(url_to_check)
```

Ova skripta šalje URL na VirusTotal i provjerava da li je označen kao maliciozan.

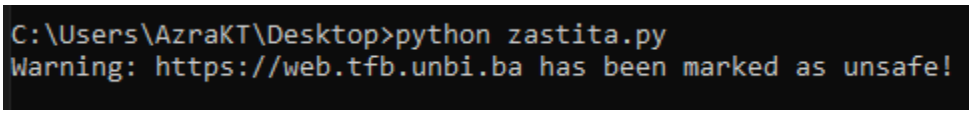
Prvobitno je potrebno prijaviti se na stranicu VirusTotal, te kopirati API za provjeru i zamijeniti ga u kôdu sa **`YOUR_VIRUSTOTAL_API_KEY.`**

Nakon toga, za testiranje je potrebno postaviti URL koji se želi provjeriti na mjesto **`url_to_check = "http://example.com".`**

Sljedeći korak je otvaranje terminala i pokretanje fajla preko naredbe:

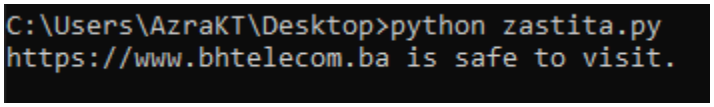
```
python NameFile.py
```

Ako je stranica prijavljena kao zlonamjerna dobije se sljedeća poruka koja je prikazana na *Slici 1.1.*



Slika 1.1. *Prikaz nakon provjere (zlonamjerna stranica)*

Ako stranica nije prijavljena kao zlonamjerna dobije se sljedeća poruka koja je prikazana *na Slici 1.2.*



Slika 1.2. *Prikaz nakon privjere (sigurna stranica)*

Sigurno korištenje interneta zahtijeva poznavanje sigurnosnih prijetnji i implementaciju odgovarajućih metoda zaštite. Korisnici mogu zaštititi svoje podatke koristeći antivirusne programe, enkripciju, sigurne mrežne prakse i provjerene alate.

2. RAČUNARSKA SIGURNOST I ZAŠTITA PODATAKA

Porastom broja krađa podataka u digitalnom obliku zaštita podataka na računaru i vanjskim uređajima za pohranu podataka postala je prioritet za mnoge firme koje u svakodnevnom poslovanju koriste povjerljive i osjetljive podatke. Gubljenje važnih informacija može imati velike financijske posljedice za neku firmu, u ekstremnim situacijama čak i kobne po poslovanje. Stoga se većina organizacija pokušava zaštiti od neovlaštenog pristupa osjetljivim podacima. U prošlosti su korištene različite metode kojima se pokušavalo osigurati tajnost i integritet važnih podataka. Mnoge od tih metoda uključivale su jednostavne algoritme.

Razvojem matematičkih nauka, tehnologije i računarstva, došlo je do pojave složenih algoritama koji su sadržavali složene matematičke proračune. Kriptovanje, ograničavanje pristupa i zaštita dokumenata iza tzv. vatrozida (eng. *firewall*) neke su od uobičajenih tehnika zaštite osjetljivih informacija. Uz takve metode koristi se i digitalno potpisivanje dokumenata i digitalni vodeni žigovi (*watermarks*). U suštini napadi se mogu definisati kao akcije koje su usmjerene na ugrožavanje sigurnosti informacija, računarskih sistema i mreža.

Postoji više vrsta napada, ali generalno gledajući mogu se klasificirati u sljedeće četiri kategorije:

1) presijecanje (napad na raspoloživost),
2) presretanje(napad na povjerljivost),
3) izmjena (napad na integritet) i
4) fabrikacija (napad na autentičnost).

Kada se govori o konkretnim primjerima veliku opasnost predstavljaju zlonamjerni programi, u koje spadaju:

1) virusi,
2) crvi,
3) trojanski konji,
4) špijunski i reklamni programi,
5) alati za dobijanje administrativnih ovlasti na sistemu (eng. *rootkit*) itd.

Pored njih tu su još neke vrste napada kao što su napad preljevom spremnika, napad uskraćivanjem usluge, fišing napad, snifing napad, itd. Treba pomenuti i osobe koje stoje iza napada, a to su zlonamjerni korisnici, danas često nazivani hakerima, iako je to dosta kontraverzan termin. Prilikom napada mogu se koristiti različitim tehnikama, od kojih je bitno izdvojiti tehnike aktivnog i pasivnog skeniranja. Postoji mnogo različitih metoda zaštite računarskih sistema, računara, mreža, te podataka i dokumenata. Mogu se podijeliti na softverske i hardverske metode.

Neke od značajnijih softverskih metoda zaštite su:

1) kriptovanje,
2) upotreba vodenih žigova, šifri, kriptografskih sažetaka,
3) korištenje digitalnih potpisa.

Pored njih kao zaštita na računaru se koriste još i beckup, antivirusna, antispyware i antibootkit rješenja. Uz navedene metode zaštite, treba pomenuti i hardverske načine zaštite. Tu je prije svega korištenje hardverskog vatrozida (postoji i softverski), hardverskog ključa itd. Još je bitno i korištenje IDS (eng. *Intrusion Detection System*), IPS (eng. *Intrusion Prevention System*) i SIEM sistema. U nastavku će biti ukratko opisana kriptografija kao metod zaštite.

3. KRIPTOGRAFIJA

Kriptografija je naučna disciplina koja se bavi proučavanjem metoda za slanje poruka u takvom obliku da ih samo onaj kome su namijenjene može pročitati. Djelotvorna komunikacija je bila vrlo bitan faktor kraljevima, kraljicama i vojskovođama stoljećima, zbog svjesnosti o posljedicama koje bi mogle rezultirati ukoliko njihove poruke dođu u krive (suparničke) ruke. Na taj način bi se otkrile zabranjene i vrlo bitne, te dragocjene tajne. Ova opasnost je potaknula razvoj šifri i kôdova koje se mogu definirati kao sredstva kojima se postiže da poruku može pročitati samo osoba kojoj je namijenjena.

Kroz godine, osnivali su se šifranski odjeli u pojedinim zemljama, a nasuprot njima i šifrolomci (razbijači šifara). Historija šifara temelji se na stogodišnjoj borbi izmedu tvoraca šifri i njihovih razbijača, pošto je svaka šifra izložena stalnim napadima protivnika. U slučaju pobjede protivnika, šifra postaje beskorisna. To je usporedivo sa hrpom bakterija u organizmu koje žive i bujaju dok medicina ne pronađe lijek ili otrov koji ih ubija. Današnji komunikacijski kanali se vrlo lako prisluškuju zbog satelita od kojih se odražava telefonski poziv, te kompjutera preko kojih protiče elektronska pošta. Današnja privatnost korisnika nije zaštićena, a izlaz iz tog problema može se pronaći u enkripciji, koja je sastavni dio kriptografije.

3.1. Osnovni pojmovi u kriptografiji

Kriptografija je naučna disciplina koja se bavi proučavanjem metoda za slanje poruka u oblicima čitljivim samo onima kojima su i namijenjene. Sama riječ kriptografija je grčkog porijekla i mogla bi se doslovno prevesti kao tajnopis. Cilj kriptografije je omogućiti nesmetano komuniciranje pošiljaoca i primaoca poruke preko nesigurnog komunikacijskog kanala tako da treća osoba ne može razumijeti njihove poruke.

Kriptografski algoritam ili šifra je matematička funkcija koja se koristi za šifriranje i dešifriranje. Općenito, radi se o dvije funkcije, jednoj za šifriranje, a drugoj za dešifriranje. Te funkcije preslikavaju osnovne elemente otvorenog teksta (najčešće su to slova, bitovi, grupe slova ili bitova) u osnovne elemente šifrata i obratno. Funkcije se biraju iz određene familije funkcija u ovisnosti o ključu. Skup svih mogućih vrijednosti ključeva naziva se *prostor ključeva*.

Enkripcija je postupak koji se primjenjuje na strani pošiljatelja. Njome se otvoreni tekst pretvara u šifrirani ili kodirani tekst pomoću odgovarajućeg ključa. Zbog toga se enkripcija dijeli na šifriranje i kodiranje. Kod šifriranja, bazična jedinica je jedno slovo, par slova (bigram), a ponekad i više slova (poligram), a kod kodiranja, bazična jedinica je riječ ili skup riječi. Zbog toga, ne postoji neka oštra teorijska granica izmedu kôdova i šifri. Šifriranje je postupak koji

se dijeli na transpoziciju (svako slovo zadržava svoj identitet, ali mijenja mjesto) i supstituciju (svako slovo zadržava svoje mjesto, ali mijenja identitet).

Dekripcija je postupak kojim se šifrirani ili kodirani tekst pretvara u otvoreni tekst pomoću unaprijed poznatog ključa i algoritma. Primjenjuje se na strani primatelja poruke. Dakle, ovaj postupak je legalan, jer primatelj komunicira sa pošiljateljem.

Samim postupkom nastaje dekriptat.

Kriptosistem se sastoji od kriptografskog algoritma ili šifre, te svih mogućih otvorenih tekstova, šifrata i ključeva. Kriptosistemi se obično klasificiraju obzirom na sljedeće kriterija:

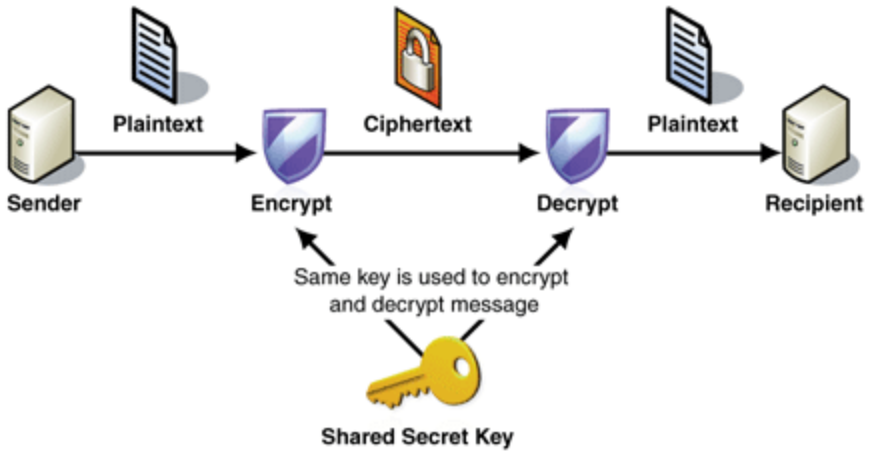
1) **Tip operacija koji se koristi pri šifriranju:**
   a) Supstitucijske šifre
   b) Transpozicijske šifre

2) **Način na koji se obrađuje otvoreni tekst:**
   a) blokovne šifre obrađuje se jedan po jedan blok elemenata otvorenog teksta koristeći jedan te isti ključ
   b) protočne šifre elementi otvorenog teksta obrađuju se jedan po jedan koristeći pri tome niz ključeva engl. *Keystream* koji se paralelno generira

3) **Tajnost i javnost ključeva**
   a) Kriptosistemi s tajnim ključem
   b) Kriptosistemi sa javnim ključem

3.2. Kriptosistemi sa tajnim ključem

Kod simetričnih, konvencionalnih kriptosistema tj. kriptosistema sa tajnim ključem, pošiljatelj i primatelj trajno izabiru ključ K pomoću kojeg generiraju funkcije za šifriranje i dešifriranje. U ovom slučaju su funkcije iste te su lake za „provaljivanje“. Iz tog razloga, sigurnost simetrilnih kriptosistema leži u tajnosti ključa, što i predstavlja veliki nedostatak, jer pošiljatelj i primatelj prije šifriranja moraju biti u mogućnosti da razmjene tajni ključ preko nekog sigurnog komunikacijskog kanala, pomoću kurira ili se osobno sresti. To je nekada teško izvedivo, naročito ako su oni na velikoj udaljenosti i ako su komunikacijski kanali koji su im na raspolaganju prilično nesigurni. Pored toga, tajni ključ se mora često mijenjati, jer šifriranje više puta istim ključem smanjuje sigurnost.

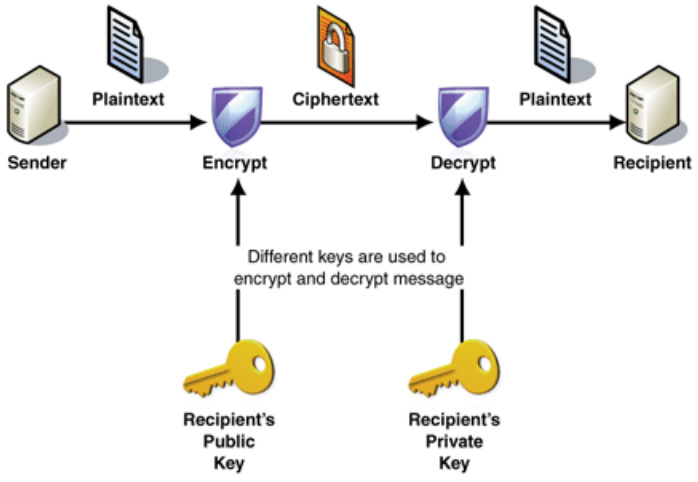


Slika 3.1. *Kriptosistem sa tajnim ključem*

Najpoznatiji kriptosistemi sa tajnim ključem su: Cezarova šifra, Vigenerova šifra, Playfairova šifra.

3.3. Kriptosistemi sa javnim ključem

Kod kriptosistema sa javnim ključem, odnosno asimetričnih kriptosistema je nemoguće, u nekom razumnom vremenu, odrediti ključ za dešifriranje (uprkos tome što je ključ za šifriranje poznat) koji je javni ključ. Naime, bilo ko može šifrirati poruku pomoću njega, ali samo osoba koja ima odgovarajući ključ za dešifriranje (privatni ili tajni ključ) može dešifrirati tu poruku. Ideju jednog takvog kriptosistema iznijeli su Whitfield Diffie i Martin Hellman, kada su dali prijedlog rješenja problema razmjenjivanja ključeva za simetrične kriptosisteme putem nesigurnih komunikacijskih kanala. U svrhu realizacije ideje kriptosistema sa javnim ključem, koriste se osobine *jednosmjerne funkcije* (f(x) lako izračunati ali f−1 jako teško, osim ukoliko je poznat neki dodatni podatak (eng. *trapdoor*- tajni ulaz)).



Slika 3.2. *Kriptosistem sa javnim ključem*

Prednosti i nedostaci u usporedbi sa simetričnim kriptosistemima:

- nema potrebe za sigurnim komunikacijskim kanalom
- kriptografija javnog ključa ne koristi se za šifriranje poruka, već za šifriranje ključeva jer su algoritmi s javnim ključem puno sporiji (oko 1000 puta) od modernih simetričnih algoritama
- kriptosistemi sa javnim ključem slabi su na napad ”odabrani otvoreni tekst”

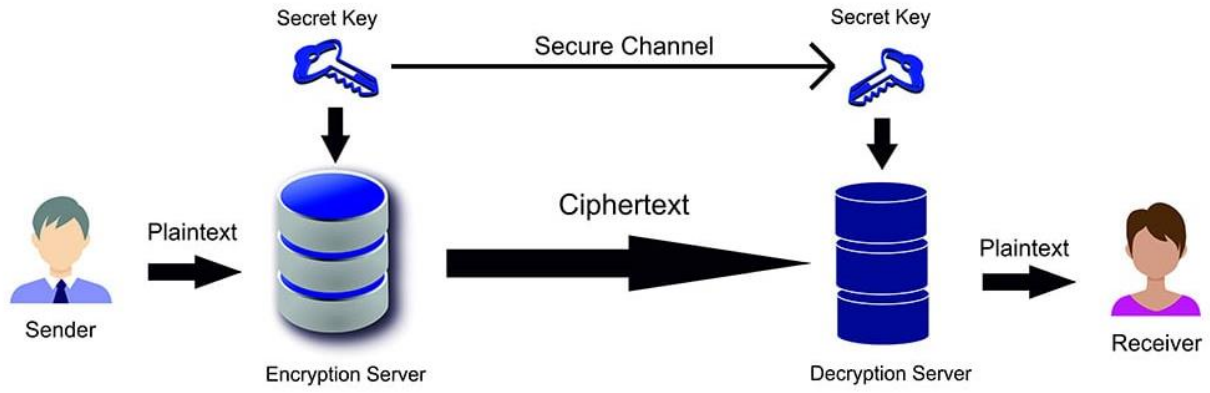
3.4. Vrste kriptografskih algoritama

Kriptografski algoritmi dijele se na:

1. **Simetrične algoritme** – isti ključ se koristi za šifriranje i dešifriranje (AES, DES, 3DES).
2. **Asimetrične algoritme** – koristi se **javni ključ** za šifriranje i **privatni ključ** za dešifriranje (RSA, ECC).
3. **Hash funkcije** – nepovratna matematička transformacija podataka u jedinstveni otisak (MD5, SHA-256, SHA-3).

3.4.1. Simetrična kriptografija

Simetrična kriptografija je tradicionalni način zaštite podataka koji se prenose od jednog do drugog učesnika u komunikaciji. Ovaj način zaštite podataka star je gotovo koliko i ljudska komunikacija. Suština simetrične kriptografije je da je za sigurnu komunikaciju neophodno da oba učesnika imaju isti ključ koji im omogućava šifriranje i dešifriranje podataka.



Slika 3.3. *Simetrična kriptografija*

U osnovi svi simetrični kriptografski algoritmi obavljaju dvije operacije supstituciju i transpoziciju. Prvi šifratori su obavljali samo jednu od ovih operacija i to samo jednom. Savremeni šifratori kombinuju ove dvije operacije ponavljajući ih više puta. Međutim, filozofija je ista. Supstitucija predstavlja zamjenu simbola drugim simbolima.

Ključne karakteristike simetrične kriptografije:

- **Jedan ključ** – isti ključ se koristi za obje operacije.
- **Brza i efikasna** – posebno pogodna za šifriranje velikih količina podataka.
- **Sigurnosni izazov** – ključ mora biti sigurno podijeljen između pošiljaoca i primaoca.

Najpoznatiji simetrični algoritmi su:

- **AES (eng. Advanced Encryption Standard)**
- **DES (eng. Data Encryption Standard)**
- **3DES (eng. Triple DES)**
- **Blowfish, Twofish, RC4**

Primjer: Matematički model simetrične kriptografije

Uz pretpostavku da postoji poruka (*plaintext*) **P**, ključ **K**, i enkripcijsku funkciju **E**.

Matematički model šifriranja izgleda ovako:

$$C = E_K(P)$$

gdje je **C** šifrirani tekst (*ciphertext*).

Za dešifriranje koristi se obrnuti proces:

$$P = D_K(C)$$

gdje je **D** funkcija dešifriranja koja koristi isti ključ **K**.

U nastavku će biti prikazan primjer *AES algoritma* za enkripciju i dekripciju koristeći *Python.*

```
from Crypto.Cipher import AES
import base64

# Function to add padding
def pad(text):
    return text + (16 - len(text) % 16) * ' '
```

```python
# The key must be 16 bytes long
key = b"1234567890123456"  # Tačno 16 bajtova


# Text to encrypt
plain_text = "Osjetljivi podaci"

# Create AES object and encrypt the text
cipher = AES.new(key, AES.MODE_ECB)
encrypted_text = cipher.encrypt(pad(plain_text).encode())

# Display encrypted text
print("Encrypted text:", base64.b64encode(encrypted_text).decode())

# Decrypt the text
decrypted_text = cipher.decrypt(encrypted_text).decode().strip()
print("Decrypted text:", decrypted_text)
```

Prikaz nakon pokretanja programa (AES algoritam):

```
C:\Users\AzraKT\Desktop>python aess.py
Encrypted text: jVvk8kTGrMWJriqnxQUjHw==
Decrypted text: Sensitive data
```

Slika 3.4. *Realizacija simetrične kriptografije*

U ovom primjeru prikazana je upotreba **AES** *(eng. Advanced Encryption Standard)* algoritma za šifriranje i dešifriranje osjetljivih podataka. AES je široko korišten simetrični šifrirajući algoritam koji osigurava povjerljivost podataka pretvaranjem običnog teksta u nečitljiv šifrovani tekst pomoću tajnog ključa.

Specifično, u ovom primjeru korišten je **ECB** *(eng. Electronic Codebook)* način rada AES-a, koji šifrira svaki blok podataka neovisno. Iako je ECB jednostavan i efikasan, nije preporučljiv za šifriranje većih količina podataka u praksi zbog sigurnosnih ranjivosti poput ponavljanja obrazaca u šifrovanom tekstu. Sigurniji načini rada, poput **CBC** *(eng. Cipher Block Chaining) ili GCM (eng. Galois/Counter Mode)*, obično se preferiraju zbog bolje sigurnosti.

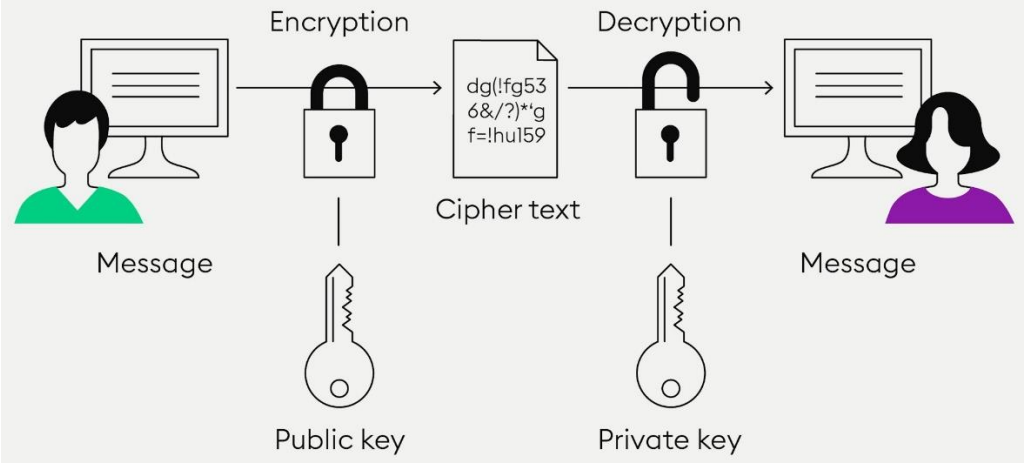
Također, šifrovani podaci prikazani su u **Base64** formatu. **Base64** je tehnika kodiranja koja omogućava predstavljanje binarnih podataka u tekstualnom formatu. Zbog prirode binarnih podataka (kao što je šifrovani tekst), koji često sadrže neprikladne znakove za tekstualne protokole (npr. specijalne karaktere ili netipične znakove), Base64 kodira te podatke u ASCII znakove (koji

su čitljivi i sigurno prenosivi kroz različite sisteme). Base64 je često korišten za prijenos binarnih podataka kroz tekstualne kanale kao što su e-mail ili web aplikacije.
Proces dešifriranja uspješno je vratio originalni obični tekst, pokazujući reverzibilnu prirodu simetrične šifriranja. Važno je napomenuti da je u stvarnim aplikacijama upravljanje ključevima ključno za održavanje sigurnosti sistema. Ključ mora biti čuvan u tajnosti, jer cijela sigurnost šifriranog sistema zavisi od njega.

Ovaj primjer predstavlja osnovnu uputu za AES šifriranje, pokazujući kako ga primijeniti za šifriranje i dešifriranje tekstualnih podataka. U praksi, može se proširiti na rad s kompleksnijim vrstama podataka, većim skupovima podataka i integrirati u šire sigurnosne sisteme za zaštitu osjetljivih informacija.

3.4.2. Asimetrična kriptografija

Razvoj savremenih elektroničkih komunikacija, a pogotovo upotreba računara i računarskih mreža učinili su problem distribucije ključeva simetrične kriptografije još većim. Računarske mreže omogućavaju brzu razmjenu podataka, ali u svojim počecima nisu pravljene da budu veoma sigurne. Činjenica da je za sigurnu upotrebu računarskih mreža bilo potrebno organizovati distribuciju ključeva na neki drugi način, te da je broj potencijalnih korisnika stalno rastao uticali su na shvatanje potrebe da se nešto promijeni. Umjesto jednog istog ključa za šifriranje i dešifriranje predloženo je postojanje para ključeva: javnog i privatnog. Javni ključ (KJ) je dostupan svima i to je ključ koji se koristi za šifriranje podataka. Odgovarajući privatni ključ (KP) je tajan i samo taj ključ omogućava dešifriranje podataka šifriranih javnim ključem iz para. Na ovaj način se rješava problem distribucije ključeva i njihovog broja. Svaki subjekat koji želi sigurnu komunikaciju može objaviti svoj javni ključ i svako ko mu želi poslati šifriranu poruku koristi taj ključ za šifriranje. Pošto jedino ovaj subjekt ima privatni ključ koji može dešifrirati podatke obezbjeđena je povjerljivost podataka. Nema potrebe za posebne kanale za sigurnu distribuciju ključeva i generisanje ključeva za svaki par koji komunicira.



Slika 3.5. *Asimetrična kriptografija*

Da bi ovakav sistem radio neophodno je obezbijediti da se na osnovu znanja javnog ključa ne može izračunati privatni ključ. Ova dva ključa moraju biti matematički povezana tako da će uvijek biti teoretski moguće izračunati jedan iz drugog, ali ovo treba učiniti računski neisplativim. Odnosno resursi potrebni za izračunavanje privatnog ključa iz javnog treba da budu nesrazmjerno veći od vrijednosti informacija koje se šifriraju. Sa druge strane potrebno je obezbijediti jednostavno i brzo šifriranje i dešifriranje. Za ovo je potrebna takozvana "jednosmjerna funkcija sa tajnom kraticom" (trap-door one-way function). Funkcija je jednosmjerna jer je lako izračunati u jednom pravcu (šifriranje), a nesrazmjerno težu u drugom (dešifriranje). "Tajna kratica" znači da je funkciju lako izračunati i u težem pravcu ako se posjeduje tajna informacija (privatni ključ).

Asimetrična kriptografija koristi **dva ključa**: **javni ključ** (public key) i **tajni ključ** (private key).

Razlikuje se od simetrične kriptografije, koja koristi isti ključ za šifriranje i dešifriranje podataka.

- **Javni ključ** se koristi za šifriranje podataka i može se slobodno dijeliti s drugima.
- **Tajni ključ** se koristi za dešifriranje podataka i mora ostati privatno i sigurno.

Značaj asimetrične kriptografije je u tome što omogućava siguran prijenos podataka čak i u nesigurnim okruženjima, jer samo primatelj s odgovarajućim privatnim ključem može dešifrirati šifrirane podatke.

3.4.2.1. RSA algoritam

RSA (*eng. Rivest-Shamir-Adleman*) je jedan od najpoznatijih asimetričnih kriptografskih algoritama. Temelji se na teoriji brojeva i koristi se za šifriranje i digitalne potpise.

Javni ključ je sastavljen od brojeva e i n, gdje je e eksponent i n produkt dvaju velikih prostih brojeva.

Tajni ključ je broj d, koji je veza između e i n, i koji se koristi za dešifriranje podataka.

RSA algoritam uključuje tri glavna koraka:

1. Generisanje ključeva
2. Šifriranje
3. Dešifriranje

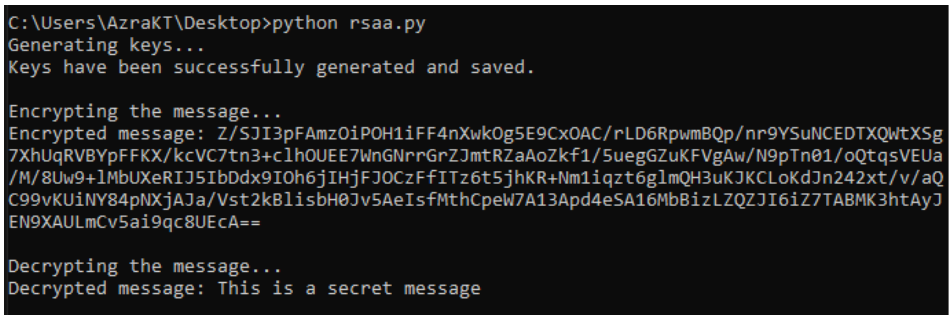
Primjer 1: Implementacija RSA algoritma u Pythonu

U nastavku će biti prikazana implementacija RSA algoritma u programskom jeziku Python. Korisniku će biti omogućeno enkripcija unešene poruke, kao i dekripcija iste, odnosno vraćanje u izvorni oblik.

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from base64 import b64encode, b64decode

# 1. Generating RSA keys
def generate_keys():
    key = RSA.generate(2048)  # Generate a 2048-bit key
    private_key = key.export_key()
    public_key = key.publickey().export_key()

    # Saving the keys to files
    with open("private.pem", "wb") as f:
        f.write(private_key)

    with open("public.pem", "wb") as f:
        f.write(public_key)

    print("Keys have been successfully generated and saved.")

# 2. Encrypting a message
def encrypt_message(plain_text):
    # Load the public key
    with open("public.pem", "rb") as f:
        public_key = RSA.import_key(f.read())

    # Create a PKCS1_OAEP encryption object
    cipher = PKCS1_OAEP.new(public_key)
```

```
    # Encrypt the message
    encrypted_message = cipher.encrypt(plain_text.encode())

    # Show the encrypted message in Base64 format
    encrypted_message_b64 = b64encode(encrypted_message).decode()
    print("Encrypted message:", encrypted_message_b64)

    return encrypted_message_b64

# 3. Decrypting the message
def decrypt_message(encrypted_message_b64):
    # Load the private key
    with open("private.pem", "rb") as f:
        private_key = RSA.import_key(f.read())

    # Create a PKCS1_OAEP decryption object
    cipher = PKCS1_OAEP.new(private_key)

    # Convert the Base64 encoded encrypted message back to binary format
    encrypted_message = b64decode(encrypted_message_b64)

    # Decrypt the message
    decrypted_message = cipher.decrypt(encrypted_message).decode()
    print("Decrypted message:", decrypted_message)

# Main program
def main():
    print("Generating keys...")
    generate_keys()  # Generate the keys

    # Encrypting the message
    message = "This is a secret message"
    print("\nEncrypting the message...")
    encrypted_message = encrypt_message(message)

    # Decrypting the message
    print("\nDecrypting the message...")
    decrypt_message(encrypted_message)

if __name__ == "__main__":
    main()
```

Prikaz nakon pokretanja programa (RSA algoritam):



Slika 3.6. *Realizacija RSA algoritma*

U ovom primjeru prikazana je upotreba **RSA** algoritma, jednog od najpoznatijih asimetričnih algoritama za šifriranje i digitalno potpisivanje. RSA se temelji na matematički složenim operacijama vezanim za proste brojeve i njihovu faktorizaciju, a osigurava povjerljivost i integritet podataka koristeći par ključeva: **javni** ključ za šifriranje i **tajni** ključ za dešifriranje.

Za razliku od simetričnih algoritama, kao što je AES, u kojima se koristi isti ključ za šifriranje i dešifriranje, RSA koristi dva ključa: jedan javni, koji je dostupan svima i služi za šifriranje podataka, i jedan tajni, koji je poznat samo primatelju i služi za dešifriranje. Ova asimetričnost omogućava sigurnost u komunikaciji bez potrebe za prethodnim dijeljenjem ključeva.

U ovom primjeru, RSA je korišten za šifriranje i dešifriranje jednostavnog teksta. **Javni ključ** se koristi za šifriranje podataka, dok **tajni ključ** omogućava dešifriranje tih podataka i vraćanje originalnog teksta. Ključevi su generirani pomoću matematičkih funkcija, a proces šifriranja i dešifriranja je uspješno implementiran.

Jedna od glavnih prednosti RSA algoritma je ta što omogućava sigurnu razmjenu podataka u nesigurnim okruženjima, kao što je internet, bez potrebe za unaprijed podijeljenim ključevima. Iako je RSA izuzetno siguran, on je sporiji od simetričnih algoritama kao što je AES, pa se često koristi za šifriranje manjih količina podataka ili za sigurno razmjenjivanje ključeva za simetričnu šifru.

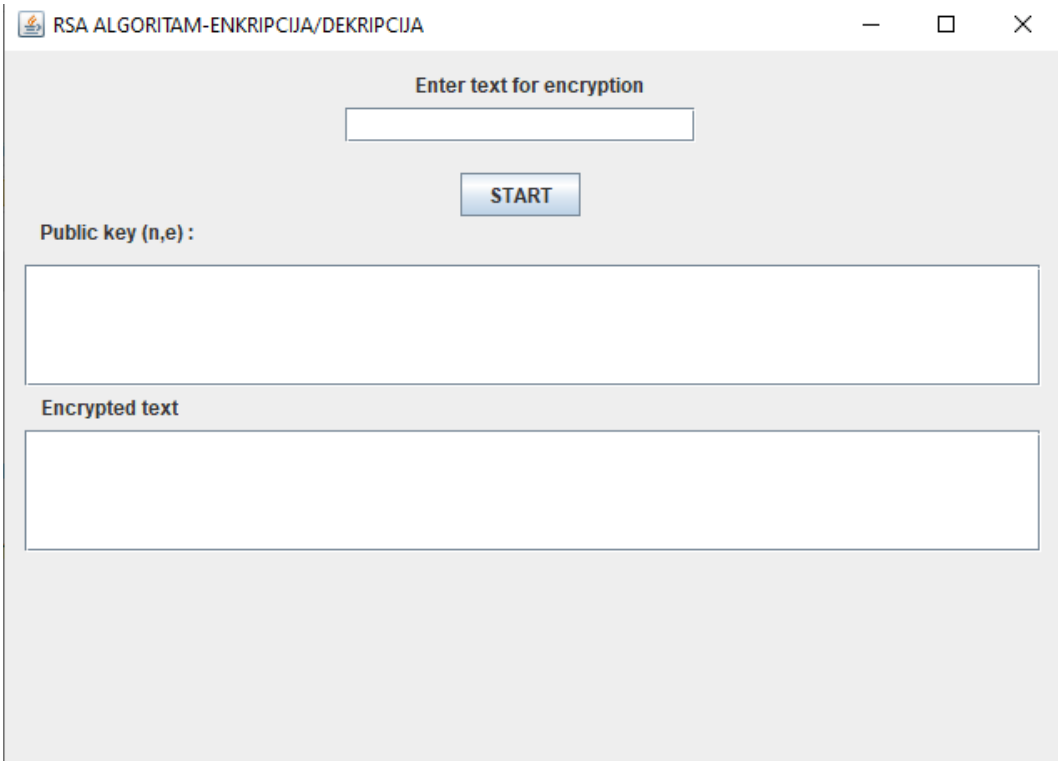
RSA se široko koristi u mnogim sigurnosnim protokolima, uključujući **SSL/TLS** za sigurne internet veze, te za digitalne potpise i autentifikaciju.

Ovaj primjer daje osnovnu uputu za upotrebu RSA algoritma za šifriranje i dešifriranje, dok u stvarnim aplikacijama može biti proširen za složenije sisteme, uključujući digitalne potpise, autentifikaciju i sigurne transakcije.

Primjer 2. Implementacija RSA algoritma u Java aplikaciji

U nastavku će biti prikazana realizacija te praktični primjer upotrebe RSA algoritma. Program se sastoji od toga da korisnik unese tekst koji želi šifrirati, nakon toga program kreira javne i privatne ključeve. Javni ključ koji je inače dostupan prikazuje se unutar samog interfejsa programa, dok se privatni ključ u ovom primjeru zapisuje u datoteku *PK.txt.* Nakon toga se unešeni tekst enkriptuje (šifrira) te se šifrirana poruka ispisuje u određeno polje unutar interfejsa. Unutar programa je predviđena i dekripcija dobivene šifrirane poruke, te se nakon iste, dobiveni rezultat ispisuje na interfejsu. Uz pretpostavku da program tačno izvrši dekripciju, dešifrovana poruka bi trebala imati isti sadržaj kao i unesena na početku samog programa.

Za kreiranje programa korišten je Apache NetBeans IDE kao i Java JDK. Za kreiranje samog UI-a korišten je Swing GUI Forms ,a za realizaciju programske logike korištena je Java.



Slika 3.7. *Prikaz interfejsa programa RSA Algoritam Test*

Klikom na dugme „Pokreni“ aktivira se triger „button1ActionPerformed“ te se realizira slijedeća funkcija:

```
private void button1ActionPerformed(java.awt.event.ActionEvent evt)
        RSA rsa = new RSA(1024);
        String tekst = this.unosTeksta.getText();
        /* Displaying the public key */
        this.jTextArea1.setText(rsa.getN() + "\n" + rsa.getE());
        BigInteger tekstUBroj = new BigInteger(tekst.getBytes());
        /* Encryption of the entered text */
        BigInteger sifriraniTekst = rsa.enkripcija(tekstUBroj);
        this.jTextArea2.setText(sifriraniTekst.toString());

        /* Decryption of the encrypted text to the original */
        tekstUBroj = rsa.dekripcija(sifriraniTekst);
        String text2 = new String(tekstUBroj.toByteArray());
        this.ispisDesifriranogTeksta.setText("Decrypted text: " + text2);
    }
```

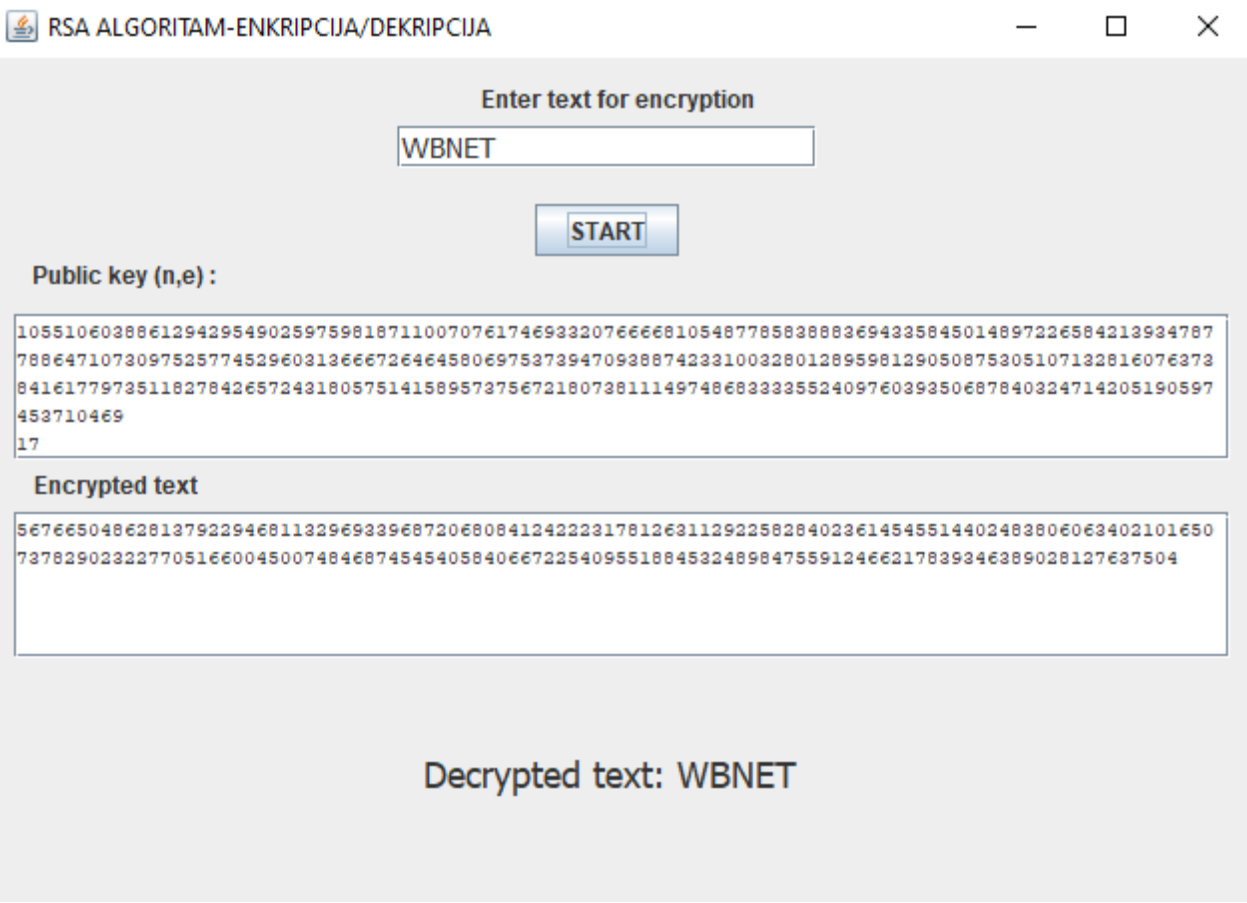
U nastavku će biti prikazana klasa RSA.java u kojoj se nalazi programska logika korištena u zadatku, te su uz komentare objašnjene funkcije korištene u programu.

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class RSA {

    private int bitlen = 1024;
    BigInteger n, d, e;

    /**
     * Creating a class instance for encryption and decryption.
     */
    public RSA(int bits) {
        bitlen = bits;
        this.generirajKljuceve();
    }

    /**
     * Generate a new set of encryption keys
     */
    private synchronized void generirajKljuceve() {
        SecureRandom r = new SecureRandom();
        BigInteger p = new BigInteger(bitlen / 2, 100, r);
        BigInteger q = new BigInteger(bitlen / 2, 100, r);
        n = p.multiply(q);
```

```
        BigInteger m = (p.subtract(BigInteger.ONE)).multiply(q
                .subtract(BigInteger.ONE));
        e = new BigInteger("17");

        /**
         * Use a fixed public key (e.g., e=17) for encryption
         */
        while (m.gcd(e).intValue() > 1) {
            e = e.add(new BigInteger("2"));
        }
        d = e.modInverse(m);


        /**
         * saving the private key to the file PK.txt (p,q,d)
         */
    pisanjePKuFajl(p.toString() + "\n" + q.toString() + "\n" + d.toString());
    }

    private static void pisanjePKuFajl(String data) {
        try {
            Files.write(Paths.get("PK.txt"), data.getBytes());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Text encryption
     */
    public synchronized BigInteger enkripcija(BigInteger tekst) {
        return tekst.modPow(e, n);
    }

    /**
     * Decryption of encrypted content
     */
    public synchronized BigInteger dekripcija(BigInteger tekst) {
        return tekst.modPow(d, n);
    }

    /**
     * Get methods for obtaining the public key. Returns p*q
   */

public synchronized BigInteger getN() {
        return n;
    }
```

```
    /**
     * Returns the public key e
     */
    public synchronized BigInteger getE() {
        return e;
    }
}
```

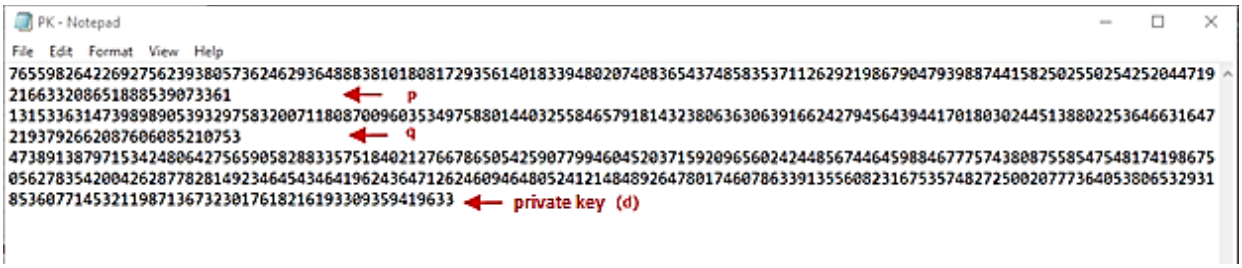
Prikaz korištenja samog programa se zasniva na tome da korisnik unese proizvoljni string koji želi da šifrira. Nakon klika na dugme „Pokreni“ program se izvršava.



Slika 3.8. *Prikaz korištenja programa RSA Algoritam Test*

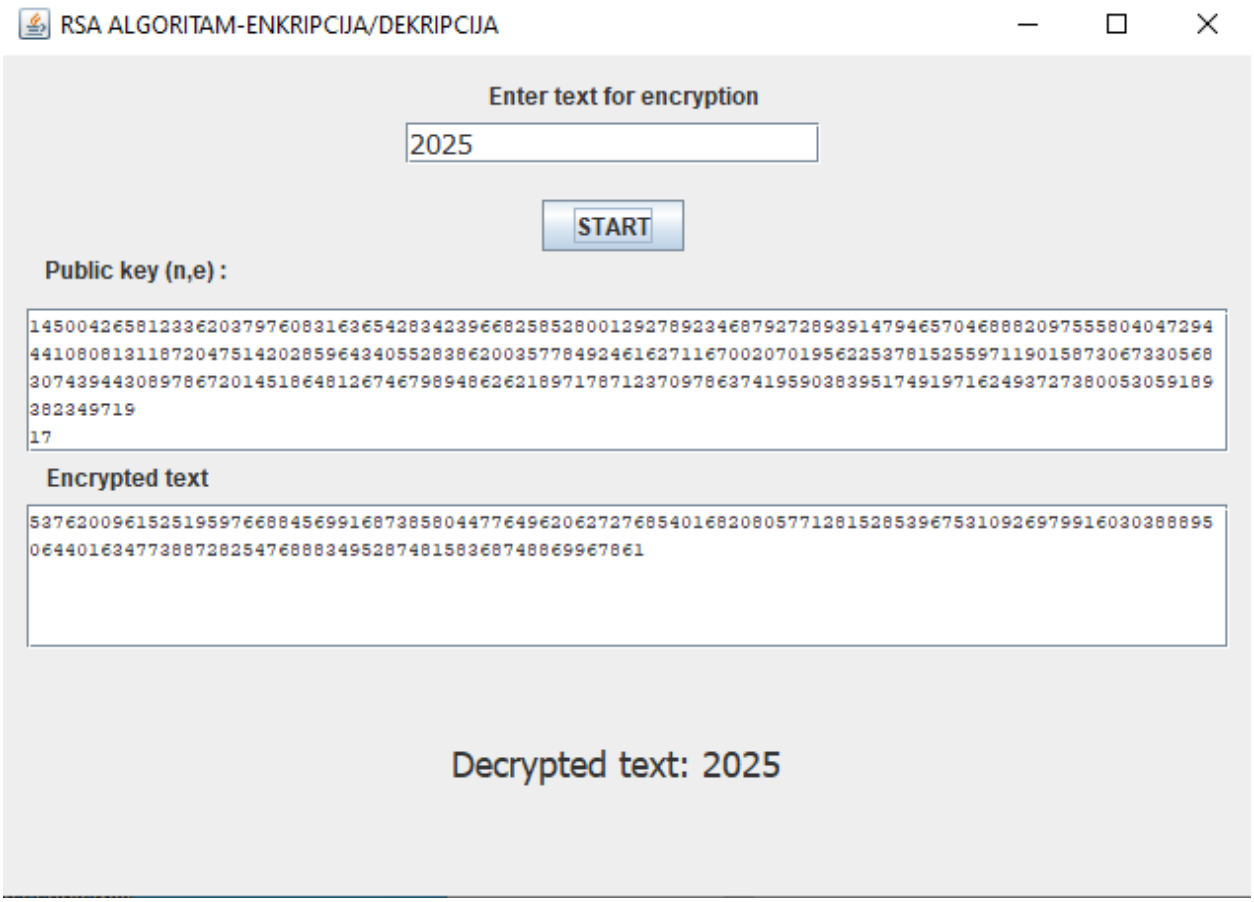
U ovome primjeru vidi se da unešena riječ „WBNET !“ klikom na dugme „START“ se izvršava program. Javno dostupni ključ je ispisan u polju Javni ključ(n,e) a s obzirom da se dešifriranjem šifrata dobije početni tekst, može se zaključiti da je program tačno izvršio zadani zadatak.

Privatni ključ je zapisan u datoteci *PK.txt*, te iako se u stvarnom svijetu ne smije javno objavljivati, u smislu demonstracije će na slici dolje biti prikazan sadržaj datoteke PK.txt.



Slika 3.8. *Prikaz datoteke PK.txt*

Prikaz rješenja šifriranja broja 22 RSA algortimom:



Slika 3.9. *Prikaz rješenja zadatka*

Ono što je bitno za napomenuti je da se u programu za javni eksponent e uzela proizvoljna vrijednost 17, zbog demonstrativne vrijednosti samog programa ,ali se svakako zbog sigurnosti samog šifriranja ne preporučuje korištenje eksponenata manjih od 65537 u komercijalnim primjenama RSA algoritma.

3.4.3. Hash funkcije

Hash funkcija je matematička funkcija koja uzima ulazne podatke (ili poruku) i generira izlaznu vrijednost fiksne dužine, koja se obično naziva "hash vrijednost" ili "hash". Hash funkcije se koriste u mnogim sigurnosnim aplikacijama, kao što su provjera integriteta podataka, digitalni potpisi, autentifikacija, pohrana lozinki, itd.

Osnovne karakteristike hash funkcija su:

1. **Jednosmjernost**: Hash funkcija je jednosmjerna, što znači da je lako izračunati hash vrijednost za neki ulaz, ali je gotovo nemoguće (ili izuzetno teško) rekonstruirati originalni ulaz iz hash vrijednosti.

2. **Fiksna dužina izlaza**: Bez obzira na veličinu ili dužinu ulaznih podataka, hash funkcija generira izlaz koji ima fiksnu dužinu. Na primjer, hash funkcija SHA-256 uvijek daje izlaz dužine 256 bita (32 bajta).

3. **Unikatnost**: Dobre hash funkcije proizvode različite hash vrijednosti za različite ulaze. Iako je matematički moguće da dva različita ulaza generiraju isti hash (to se zove sudar), kvalitetne hash funkcije čine sudare izuzetno rijetkim.

4. **Osjetljivost na promjene**: Ako se samo jedan bit u ulaznim podacima promijeni, hash vrijednost će se drastično promijeniti, što čini hash funkcije korisnim za provjeru integriteta podataka.

Popularne hash funkcije uključuju:

- **MD5** (*eng. Message Digest Algorithm* 5) – sada smatra zastarjelim zbog sigurnosnih ranjivosti.
- **SHA-1** (*eng. Secure Hash Algorithm* 1) – također se smatra nesigurnim zbog mogućnosti sudara.
- **SHA-256** – dio SHA-2 porodice, vrlo siguran i široko korišten.
- **SHA-3** – najnovija verzija SHA hash funkcija.

Primjer: Implementacija SHA-256 hash funkcije

```
import hashlib

# Input text
input_text = "Sensitive data"

# Create SHA-256 hash object
sha256_hash = hashlib.sha256()

# Update the hash object with the input data
sha256_hash.update(input_text.encode())

# Get the hash value
hashed_text = sha256_hash.hexdigest()

# Displaying the original text and its SHA-256 hash
print("Original text:", input_text)
print("SHA-256 Hash value:", hashed_text)
```

Prikaz rezultata:

```
C:\Users\AzraKT\Desktop>python hash.py
Original text: Sensitive data
SHA-256 Hash value: 474de7276ecc120b83bc42291a96a36c648a0f6428bf7c88bfdccf32bfbb4339
```

Slika 3.10. *Realizacija SHA-256 hash funkcije*

Hash funkcije, poput SHA-256, igraju ključnu ulogu u modernoj kriptografiji i sigurnosti podataka. One se koriste za generisanje izlazne vrednosti fiksne veličine (hash) na osnovu ulaznih podataka promenljive dužine, što čini proces obrnute konverzije vrlo teškim. Ključne osobine kriptografskih hash funkcija uključuju determinističnost (isti ulaz uvek proizvodi isti izlaz), brzo računanje i otpornost na sudar (teško je pronaći dva različita ulaza koja daju isti hash).

U prikazanom primjeru, koristili smo Python-ovu biblioteku hashlib kako bismo primenili SHA-256 algoritam na uzorak teksta. Dobijeni hash je jedinstven za te podatke. Pošto su hash funkcije jednosmjerne, izuzetno je teško povratiti originalne podatke iz samog hasha. Ova osobina čini hash funkcije idealnim za primene kao što su skladištenje lozinki, verifikacija integriteta podataka i digitalni potpisi.

Hash funkcije poput SHA-256 su neophodan alat za osiguranje integriteta i sigurnosti podataka, te igraju ključnu ulogu u sigurnoj komunikaciji i procesima verifikacije podataka.

3.4.4. RNG generatori

Generatori slučajnih brojeva (RNG, eng. *Random Number Generators*) ključna su komponenta u mnogim aspektima sigurnosti računarskih sustava i kriptografije. Služe za generiranje brojeva koji su, prema definiciji, nasumični i nepredvidivi. RNG-ovi su temelj za stvaranje sigurnih ključeva, digitalnih potpisa, autentifikacije i mnogih drugih operacija u savremenim informacijskim tehnologijama. S obzirom na važnost njihove uloge, kvaliteta i sigurnost RNG-a direktno utječu na cijeli sistem u kojem se koriste.

Postoje dva osnovna tipa RNG generatora:

1. **Pseudo-slučajni generatori (PRNG)** i
2. **Pravi slučajni generatori (TRNG)**.

Kvaliteta RNG-a igra ključnu ulogu u sigurnosti cijelog sistema. Ako RNG nije dovoljno dobar, može doći do predvidljivosti u generiranim brojevima, što može dovesti do ozbiljnih sigurnosnih prijetnji. Na primjer:

- *Niska entropija*: Ako generator ne koristi dovoljno nesigurnih izvora (npr. fizičkih procesa), brojevi koje generira mogu biti previše predvidljivi, čime bi napadači mogli razbiti enkripcijske ključeve.

- *Napadi na sjemenski prostor*: Ako se "sjeme" RNG-a može predvidjeti ili ponovno izračunati (npr. kod PRNG-a), napadači mogu rekonstruirati cijeli niz nasumičnih brojeva, čime bi kompromitirali sigurnost.

Iz tih razloga, kriptografski sistemi često koriste napredne tehnike poput *kriptografski sigurnih PRNG-ova* (eng. *cryptographically secure pseudo-random number generators*, CSPRNG) koji kombiniraju visoku entropiju i zaštitu od predvidljivosti. Neki od najpoznatijih CSPRNG-a uključuju algoritme poput *Yarrow* i *Fortuna*.

RNG generatori su temelj za sigurnost u kriptografiji i mnogim drugim sigurnosnim sistemima. Dok pseudo-slučajni generatori nude brzinu i efikasnost, pravi slučajni generatori pružaju višu razinu sigurnosti zbog njihove temeljne nesigurnosti. Kako bi se održala sigurnost sistema, RNG mora biti pažljivo dizajniran i implementiran, s posebnim naglaskom na zaštitu od predvidljivosti i napada. S obzirom na njegovu ključnu ulogu u generiranju kriptografskih ključeva, digitalnih potpisa i drugih sigurnosnih elemenata, visokokvalitetni RNG generatori su neophodni za osiguranje povjerenja u modernim informacijskim sistemima.

3.4.4.1. Pseudo-slučajni generatori (PRNG)

Pseudo-slučajni brojevi nisu "pravi" slučajni brojevi, već su generirani pomoću determinističkih algoritama. To znači da, iako se ponašaju kao slučajni brojevi, oni su zapravo predvidljivi ako se zna početni uslov ili "sjeme" (eng. *seed*). Pseudo-slučajni generatori široko se koriste u računarskim sistemima zbog svoje brzine i učinkovitosti.

Jedan od najpoznatijih PRNG-a je *Mersenne Twister*. Iako vrlo brz i učinkovit, PRNG-ovi nisu pogodna opcija za sigurnosne aplikacije jer ako napadači dođu do sjemena, mogu predvidjeti cijeli niz generiranih brojeva.

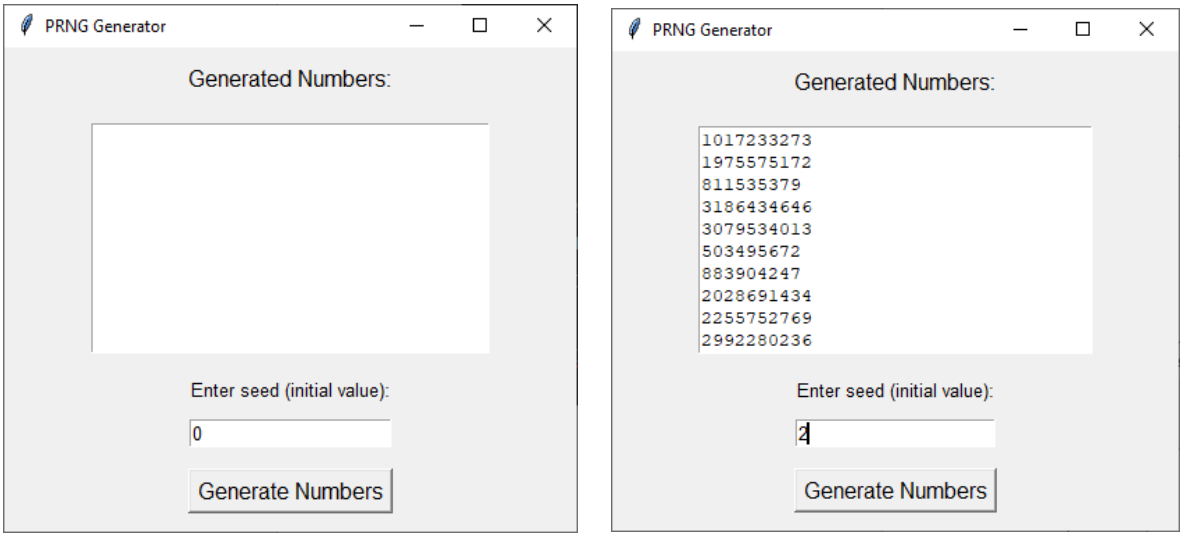
Primjer: Implementacija PRNG-a u Pythonu

U ovom program se implementira pseudo-slučajni generator brojeva (PRNG) koristeći Linear Congruential Generator (LCG) algoritam. Korisnik unosi početnu vrijednost (sjeme) u tekstualno polje, a pritiskom na dugme "Generiraj brojeve" generira se 10 nasumičnih brojeva temeljenih na tom sjemenu. Generirani brojevi se prikazuju u tekstualnom okviru u GUI aplikaciji izrađenoj pomoću *Tkinter* biblioteke. Program omogućava jednostavno generiranje različitih nizova brojeva jednostavnom promjenom početne vrijednosti.

```
import tkinter as tk
from tkinter import messagebox

# PRNG - Linear Congruential Generator (LCG)
class PRNG:
    def __init__(self, seed=0):
        self.a = 1664525  # Multiplier
        self.c = 1013904223  # Increment
        self.m = 2**32  # Modulus
        self.state = seed

    def generate(self):
        self.state = (self.a * self.state + self.c) % self.m
        return self.state

# GUI Application
class PRNGApp:
    def __init__(self, root):
        self.root = root
        self.root.title("PRNG Generator")
        self.root.geometry("400x300")
```

```
        self.prng = PRNG()  # Create an instance of PRNG

        # Label to display generated numbers
        self.output_label = tk.Label(root, text="Generated Numbers:", font=("Arial",
12))
        self.output_label.pack(pady=10)

        # Text box to display the numbers
        self.number_display = tk.Text(root, height=10, width=35)
        self.number_display.pack(pady=10)

        # Seed input field
        self.seed_label = tk.Label(root, text="Enter seed (initial value):",
font=("Arial", 10))
        self.seed_label.pack(pady=5)

        self.seed_entry = tk.Entry(root, font=("Arial", 10))
        self.seed_entry.pack(pady=5)
        self.seed_entry.insert(0, "0")  # Default value for seed

        # Button to generate numbers
        self.generate_button = tk.Button(root, text="Generate Numbers",
font=("Arial", 12), command=self.generate_numbers)
        self.generate_button.pack(pady=10)

    def generate_numbers(self):
        try:
            seed = int(self.seed_entry.get())  # Get the seed value from the input
            self.prng = PRNG(seed)  # Create a new PRNG with the new seed
        except ValueError:
            messagebox.showerror("Error", "Please enter a valid seed.")
            return

        self.number_display.delete(1.0, tk.END)  # Clear previous numbers
        for _ in range(10):  # Generate 10 numbers
            random_number = self.prng.generate()
            self.number_display.insert(tk.END, f"{random_number}\n")

# Create the main Tkinter window for the application
if __name__ == "__main__":
    root = tk.Tk()
    app = PRNGApp(root)
    root.mainloop()
```

Prikaz dobijenih rezultata:



Slika 3.11. *Izgled i testiranje aplikacije PRNG*

Kao što se može vidjeti sa Slike X, ova aplikacija omogućava korisnicima da generiraju pseudo-slučajne brojeve koristeći Linear Congruential Generator (LCG) algoritam. Unosom početne vrijednosti (sjemena) i pritiskom na dugme, aplikacija generira niz nasumičnih brojeva koji se prikazuju u grafičkom sučelju. Time se demonstrira princip rada PRNG-a, omogućujući korisnicima da istraže kako različita sjemena utječu na generirane brojeve. Aplikacija pruža jednostavan način za vizualizaciju pseudo-slučajnog procesa u stvarnom vremenu.

3.4.4.2. Pravi slučajni generatori (TRNG)

Pravi slučajni generatori temelje se na fizičkim procesima, kao što su elektronički šum ili radioaktivni raspad, kako bi generirali nepredvidljive brojeve. Ovi generatori su u pravilu sigurniji jer koriste izvor nesigurnosti koji nije deterministički i koji se ne može lako predvidjeti. Međutim, TRNG generatori su često sporiji i skuplji od PRNG-a, te se rjeđe koriste zbog zahtjeva za specifičnim hardverom.

Primjer: Implementacija TRNG-a u Pythonu

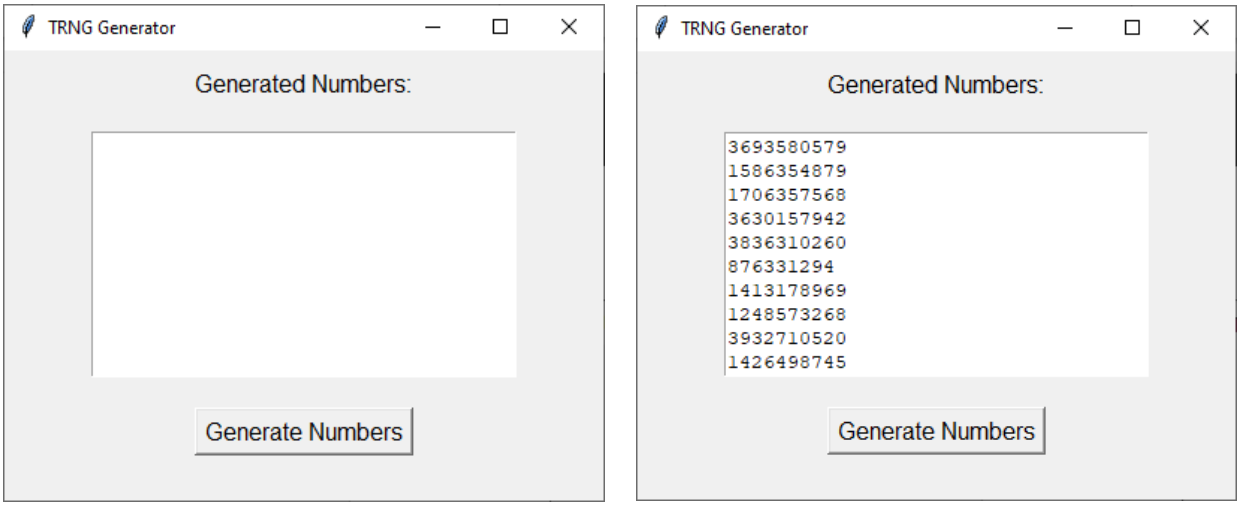
Primjer implementira aplikaciju koja generira *pravi nasumični broj* koristeći entropiju operativnog sistema, što znači da brojevi koji se generiraju nisu deterministički kao kod PRNG-a, već se temelje na stvarnom nasumičnom procesu iz računarskog sistema. Pritiskom na dugme za

generisanje brojeva, aplikacija prikazuje 10 nasumičnih brojeva na ekranu. Ova aplikacija koristi *grafičko sučelje* (GUI) za interakciju s korisnikom, omogućavajući jednostavno generiranje i prikazivanje nasumičnih brojeva u stvarnom vremenu.

```
import tkinter as tk
from tkinter import messagebox
import os

# TRNG (True Random Number Generator) using os.urandom()
class TRNG:
    def generate(self):
        # Using os.urandom() which uses entropy from the operating system to generate
random numbers
        random_bytes = os.urandom(4)  # Generate 4 bytes of random data
        random_number = int.from_bytes(random_bytes, "big")  # Convert bytes into an
integer
        return random_number

# GUI Application
class TRNGApp:
    def __init__(self, root):
        self.root = root
        self.root.title("TRNG Generator")
        self.root.geometry("400x300")

        self.trng = TRNG()  # Create an instance of TRNG

        # Label to display generated numbers
        self.output_label = tk.Label(root, text="Generated Numbers:", font=("Arial",
12))
        self.output_label.pack(pady=10)

        # Text box to display the numbers
        self.number_display = tk.Text(root, height=10, width=35)
        self.number_display.pack(pady=10)

        # Button to generate numbers
        self.generate_button = tk.Button(root, text="Generate Numbers",
font=("Arial", 12), command=self.generate_numbers)
        self.generate_button.pack(pady=10)

    def generate_numbers(self):
        self.number_display.delete(1.0, tk.END)  # Clear previous numbers
        for _ in range(10):  # Generate 10 numbers
            random_number = self.trng.generate()
```

```
            self.number_display.insert(tk.END, f"{random_number}\n")

# Create the main Tkinter window for the application
if __name__ == "__main__":
    root = tk.Tk()
    app = TRNGApp(root)
    root.mainloop()
```

Prikaz dobijenih rezultata.



Slika 3.12. *Izgled i testiranje aplikacije TRNG*

Kao što se može vidjeti sa Slike Y, aplikacija omogućava korisnicima generiranje *pravih nasumičnih brojeva*, a način generiranje istih, temeljen na entropiji operativnog sistema, garantira da generirani brojevi nisu predvidivi.

3.4.4.3. Primjena RNG generatora u kriptografiji

RNG generatori imaju ključnu ulogu u kriptografiji, gdje sigurnost sistema u velikoj mjeri ovisi o stvaranju nasumičnih brojeva koji su teški za predvidjeti. Glavne primjene RNG-a u kriptografiji uključuju:

Generiranje kriptografskih ključeva

U simetričnoj i asimetričnoj kriptografiji, ključevi koji se koriste za enkripciju i dekripciju podataka trebaju biti nasumični i teško predvidljivi. U simetričnoj kriptografiji, nasumični ključevi

koji se koriste u algoritmima poput AES-a moraju biti proizvedeni pomoću sigurnih RNG-a. Slično tome, u asimetričnoj kriptografiji, RNG se koristi za generiranje privatnih ključeva u sistemima poput RSA, ECC (Elliptic Curve Cryptography) ili DSA (Digital Signature Algorithm).

Salt i IV (Inicijalizacijski Vektor)

U mnogim enkripcijskim sistemima, poput AES u CBC modu, koristi se inicijalizacijski vektor (IV) kako bi se osigurala varijabilnost kod enkripcije istih podataka s istim ključem. IV mora biti nasumičan, a RNG generira ovaj vektor kako bi osigurao sigurnost. Također, pri pohranjivanju lozinki, RNG se koristi za generiranje "salt" vrijednosti koja se dodaje lozinci prije njezine obrade kroz hash funkciju. Ovaj dodatni sloj zaštite otežava napadačima izvođenje napada temeljenih na predbilježbama lozinki (npr. rainbow table attack).

Digitalni Potpisi i Certifikati

U digitalnim potpisima, RNG se koristi za generiranje nasumičnih brojeva koji služe u procesu potpisivanja. Osim toga, RNG je neophodan u postupku generiranja certifikata u sistemima kao što su SSL/TLS. Certifikati i potpisivanje ovih certifikata temelje se na nasumičnim brojevima koji garantiraju sigurnost u komunikacijama.

Generiranje Session Ključeva

RNG također ima važnu ulogu u protokolima poput TLS-a, gdje se koristi za generiranje session ključeva tokom sigurnih razmjena podataka. Ovi ključevi omogućuju enkripciju komunikacije između dvije strane, a njihova nasumičnost ključna je za zaštitu od napada.

ZAKLJUČAK

Završetkom ovog kursa, polaznici će steći sveobuhvatno znanje iz oblasti internet i računarske sigurnosti, uključujući zaštitu korisnika i računara na internetu, prepoznavanje i prevenciju sigurnosnih prijetnji, kao i implementaciju zaštite putem kriptografskih tehnika. Kroz praktične primjere u Pythonu, polaznicima će biti omogućeno da stečena teorijska znanja primijene u realnim situacijama, čime će razviti potrebne vještine za implementaciju sigurnosnih rješenja u digitalnom okruženju. Savladavanje kriptografskih algoritama i metoda zaštite podataka doprinosi njihovoj sposobnosti da prepoznaju i adresiraju sigurnosne izazove, čineći ih spremnim za suočavanje sa stalnim prijetnjama u današnjem dinamičnom internetu.